Learning Vision-based Reactive Policies for Obstacle Avoidance

Elie Aljalbout, Ji Chen, Konstantin Ritt, Maximilian Ulmer, and Sami Haddadin
Munich School of Robotics and Machine Intelligence
Technical University of Munich
{`name.lastname@tum.de`}

Abstract: In this paper, we address the problem of vision-based obstacle avoidance for robotic manipulators. This topic poses challenges for both perception and motion generation. While most work in the field aims at improving one of those aspects, we provide a unified framework for approaching this problem. The main goal of this framework is to connect perception and motion by identifying the relationship between the visual input and the corresponding motion representation. To this end, we propose a method for learning reactive obstacle avoidance policies. We evaluate our method on goal-reaching tasks for single and multiple obstacles scenarios. We show the ability of the proposed method to efficiently learn stable obstacle avoidance strategies at a high success rate, while maintaining closed-loop responsiveness required for critical applications like human-robot interaction.

Keywords: Learning Obstacle Avoidance, Robot Manipulators, Robot Vision

1 Introduction

During task execution, robots should be capable of operating in their workspaces without colliding with obstacles. In well-structured environments, this constraint can be easily ensured by carefully designing collision-free motion trajectories based on the understanding of the robot surroundings. In contrast, unstructured environments present the challenge of autonomously reacting to previously unknown settings. To tackle this challenge, extra efforts are needed in order to design proper perception systems, capable of understanding the environment, as well as reactive strategies to avoid the obstacles. In this work, we are concerned with obstacle avoidance for robot manipulators. In addition to the previously mentioned challenges, such systems impose additional constraints such as joint limits, singularities and self-collision. All of these aspects add to the complexity of the problem, and require proper care in the formulation of both classical and learning-based methods.

In this context, proprioceptive robot sensors enable collision detection and early reactions which can prevent substantial damages to the robot and its environment [1]. On the other hand, exteroceptive sensing (e.g. cameras) provides geometric information and objects poses which allow when required, to avoid the obstacles in the first place. This motivates vision-based motion generation and planning for obstacle avoidance. Although this problem seems trivial at first glance, it entails a multitude of challenges. Namely, most approaches in this field are based on the classical perception and planning pipeline [2, 3]. Every module in such a pipeline introduces extra errors to the overall system. Namely, state-of-the-art pose detection methods (such as DeepIM [4] and Pix2Pose [5]) still lack the required accuracy for such tasks. Similarly, real-time generation of reactive motion trajectories could be error prone, especially when the given obstacle poses are faulty.

One way to avoid the aforementioned problems is to learn model-free reactive motion policies. This can be simply done using end-to-end deep reinforcement learning (DRL). However, these methods could lack the required generalization capabilities which defies the original purpose. Additionally, DRL algorithms require long interaction times, which are time and resource expensive and could also be damaging to the robot, unless safe exploration is guaranteed [6, 7]. Furthermore, with the lack of proper action space design, DRL methods tend to fail in real-world robot learning tasks [8]. Instead, it is desirable to combine DRL with classical methods as a trade-off between sample-efficiency, interpretability, modelling efforts and modelling errors.

In this work, we adopt the latter approach. Our main goal is to relate the visual information to the robot's motion. To do that, we learn a residual strategy to react to obstacles in the robot's surroundings based on vision. This policy augments another simple hand-crafted one. Both are designed based on Riemannian Motion Policies (RMP) [9], which enables us to effectively combine them, while keeping desirable constraints such as joint limits. To further reduce the sample complexity of our method, we learn a visual latent model for images. This model enables us to reduce the complexity of our policies and also to have an interpretable and generalizable representation of the environment. Our contributions can be summarized as follows:

- We present a unified framework for learning vision-based obstacle avoidance with minimum expert knowledge.
- Our method can learn successful single obstacle avoidance strategies within less than an hour of real-time interactions with the environment, using a single manipulator.
- Our method enables simultaneous avoidance of multiple obstacles at a high success rate.
- To our knowledge, our method is the first to use DRL for vision-based obstacle avoidance on robot manipulators.

2 Related Work

Obstacle Avoidance is a fundamental problem in robotics for which many solutions have been proposed. Possible solutions can be categorized into two main groups: local and global methods [10]. Local algorithms, such as potential fields [11], only adapt the manipulator's behavior in the presence of obstacles. They are reactive closed-loop control techniques which generate local motion. Due to their low complexity, these algorithms are able to run in the inner control loop of the robot [12] which ensures the level of responsiveness that is needed for critical scenarios, such as safe human-robot interaction. However, these approaches assume Euclidean geometry in the task space of the manipulator [13] and only use internal geometry of the kinematic chain [14]. Despite their simplicity and versatility, purely reactive methods often culminate in undesirable behaviors such as instability or oscillation [9]. On the other hand, global planning-based, differential geometric approaches aim to model the intrinsically non-euclidean task space of the robot [15]. This thread of research has seen significant progress recently and primarily uses optimization to generate nonlinear trajectories [16]. Nonetheless, planning-based method are notoriously computationally expensive and lack the reactiveness of local methods. To avoid obstacles, these methods require knowledge about obstacles (e.g. keypoints) which is crucial for autonomous applications in unstructured, dynamic environments. This need for semantic understanding of the environment has sprouted a variety of research on vision-based obstacle avoidance, which is especially prominent in mobile robotics [17].

Motion Generation. Movement primitives (MP) are a common and versatile method to encode motion as a sequence of unit actions. Popular examples for this paradigm are Dynamic Movement Primitives (DMP) [18] and Probabilistic Movement Primitives (ProMP) [19]. DMPs encode motion as second-order differential equations. They can scale motion both in space and time, and enable learning by demonstration in their formulation [18]. In contrast, ProMPs encode motion primitives by a distribution over trajectories which allows the use of powerful statistical methods, such as conditioning a certain trajectory on some desired behavior. Similar to RMPs, some motion primitives model motion as a second-order differential equation, however, they are lacking the modularity and combinatorial capabilities of operational space closed-loop control [13].

Reactive Planning. Responsiveness is a crucial factor in robotic manipulation tasks with a high degree of uncertainty due to incomplete modelling, inaccurate sensors or incomplete observability [2]. Combining local reactive control and global collision-free motion planning into a unified *reactive planning* framework is common in mobile robotics [20, 21], however only a few approaches are specifically designed for high degrees of freedom (DoF) manipulators [18, 22, 3, 9]. In this work, we follow the route of a hybrid approach combining the reactive and planning capabilities of RMPs with the flexibility of DRL.

Reinforcement Learning for robotics and continuous control has seen substantial progress in recent years [23]. DRL has the potential to deliver on the promise of closing the loop between perception and motion, without the manual design of intricate features or behaviors [24]. Similar to the case of classical vision-based obstacle avoidance methods, there is a comprehensive body of literature focused on mobile robots and navigation using RL [25, 26, 27]. Only few work has been done in this area when it comes to robot manipulators, however, while keeping vision out of the loop and still assuming explicit a priori knowledge about obstacles [28]. To this end, we combine a vision-based reactive policy with a hand-designed, task-specific baseline policy, to leverage the advantages of model-free RL for reactive obstacle avoidance. We train the reactive policy to augment the baseline based on residual RL [29, 30]. By combining learning and control, our approach is robust to common pitfalls of reinforcement learning, such as stability, long-horizon and sparse-rewards.

In summary, vision-based obstacle avoidance is a highly researched topic in robotics. For navigation, many approaches have been presented to improve the accuracy, robustness and overall performance [17, 25, 27]. In general, such methods are not directly transferable to high-DoF manipulators. While the problems seem similar at a high level, they are fundamentally different. In mobile navigation the geometry of the robot is preserved throughout execution. This is not the case for articulated manipulators. This aspect introduces extra concerns such as self-collision, joint-limit avoidance, or various motion and torque/force constraints. Methods focused on high-DoF manipulators assume perfect knowledge about the obstacle position and are based on classical robotic pipelines separating vision and planning [22, 9, 18], which usually results in reduced performance [31], or rely on kinesthetic demonstrations [3]. In contrast, our approach does not make any expensive assumptions nor does it separate vision from planning, rather treats the two in a unified formalism.

3 Background

The most salient attributes of robotic manipulators are best expressed in terms of differential geometry [32]. More specifically, motion generation and control can be seen as the problem of transforming desired behavior from one or multiple smooth manifolds – representing the task space $\mathcal{T}$ – to another smooth manifold $\mathcal{C}$, the configuration space. These manifolds are related by a differential map $\phi : \mathcal{C} \to \mathcal{T}$, called the task map. By equipping these manifolds with a Riemannian metric $\mathbf{M}$, we can elegantly relate notions like angles and distances to design a curve $q(t) \in \mathcal{C}$ which implements the desired behavior of $\mathcal{T}$.

Riemannian Motion Policies. An RMP is a motion representation defined on a Riemannian manifold. The generation of this type of policies depends not only on the potential field existing in the space, but also on the curvature of the space itself [9]. Consider an arbitrary m-dimensional manifold $\mathcal{M}$ with generalized coordinates $\mathbf{x} \in \mathbb{R}^m$. In its canonical form, an RMP is defined as $(\mathbf{a}, \mathbf{M})_{\mathcal{M}}$, where $\mathbf{a} : \mathbf{x}, \dot{\mathbf{x}} \mapsto \ddot{\mathbf{x}}^d \in \mathbb{R}^m$ represents a second-order dynamical system mapping $\mathbf{x}$ and $\dot{\mathbf{x}}$ to desired accelerations $\ddot{\mathbf{x}}^d$, and $\mathbf{M} = \mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) \in \mathbb{R}^{m\times m}$ is a Riemannian metric which varies smoothly with the state $(\mathbf{x}, \dot{\mathbf{x}})$. We can interpret $\mathbf{M}$ as an inertial matrix which also defines the weight of the RMP when combined with others. In its natural form, an RMP is defined as $(\mathbf{f}, \mathbf{M})_{\mathcal{M}}$, where $\mathbf{f}$ indicates the map from position and velocity to the desired force. The force map $\mathbf{f}$ in the natural form and the acceleration map in the canonical form has the relation $\mathbf{f} = \mathbf{Ma}$. The natural form of an RMP is commonly used when space transformations are to be applied, due to the lower computational complexity of such operations using this form.

The RMP framework also provides `push`, `pull` and `add` operators. `push` and `pull` can transform an RMP defined in a certain task space into another, based on the task map ϕ and its Jacobian $\mathbf{J}$. As for `add`, it is used to compose RMPs defined in the same task space into one policy.

RMPflow is a policy synthesis framework based on [14]. In this method, RMP-tree is introduced as a tree structured computational graph. Each node in such a tree represents a Riemannian manifold and is equipped with an RMP. The root node of the tree describes the configuration space $\mathcal{C}$ of the robot where the global joint space policy is defined. The leaf nodes represent task spaces, where the designed or learned local policies are defined.

The RMPs in this framework are characterized by a set of geometric dynamical systems (GDSs). Consider a manifold $\mathcal{M}$ with generalized coordinate $\mathbf{x} \in \mathbb{R}^m$, a GDS defined on such manifold can be expressed as:

$$(\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \Xi_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}))\ddot{\mathbf{x}} + \xi_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) = -\nabla\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}, \tag{1}$$

where $\mathbf{G} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ is referred to as the metric matrix, $\mathbf{B} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ as the damping matrix and $\Phi : \mathbb{R}^m \to \mathbb{R}$ as the potential function. Additionally, $\Xi_{\mathbf{G}} := \frac{1}{2}\sum_{i=1}^{m} \dot{x}_i \partial_{\dot{\mathbf{x}}} \mathbf{g}_i$

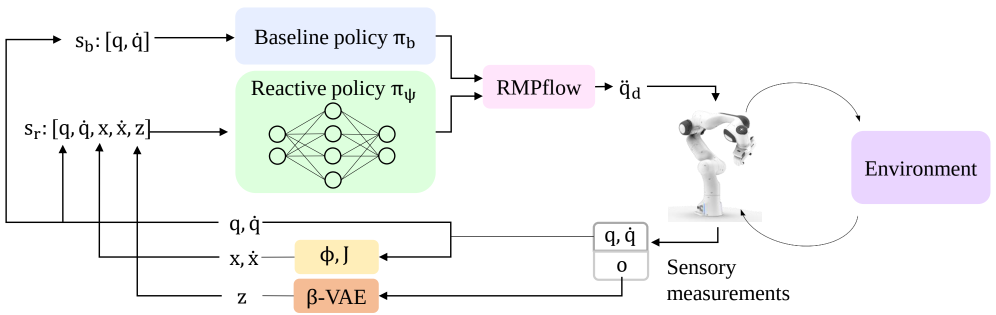


Figure 1: System Overview: At a certain time step, the system receives a visual input **o**, joint angle measurements **q** and joint velocities $\dot{\mathbf{q}}$. The image **o** is then encoded using a β-VAE encoder to produce the visual latent **z**. Using the robot kinematics ϕ and jacobian **J**, we obtain the end-effector position **x** and velocity $\dot{\mathbf{x}}$ from **q** and $\dot{\mathbf{q}}$ respectively. Subsequently **q** and $\dot{\mathbf{q}}$ are fed into a baseline policy to produce a user-defined behavior. Simultaneously, we feed all available information s_r into a reactive policy π_ψ. The latter produces a reactive behavior dependent on the objects in the environment. Both outputs are then composed together based on the RMPflow framework, to produce a desired joint acceleration $\ddot{\mathbf{q}}_d$. This acceleration is then fed into the robot controller.

and $\xi_{\mathbf{G}} := \overset{\mathbf{x}}{\mathbf{G}}\dot{\mathbf{x}} - \frac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{x}}^{\mathsf{T}}\mathbf{G}\dot{\mathbf{x}})$ are curvature terms induced by the metric **G**, where $\mathbf{g}_i$ denotes the i-th column of **G** and $\overset{\mathbf{x}}{\mathbf{G}} := [\partial_{\mathbf{x}}\mathbf{g}_i\dot{\mathbf{x}}]_{i=1}^{m}$. Based on the GDS formulation, the RMP metric term **M** is defined as $\mathbf{M} := \mathbf{G} + \Xi_{\mathbf{G}}$. The forcing term of RMP can be calculated by moving the curvature term $\xi_{\mathbf{G}}$ to the right hand side of equation 1, we then obtain:

$$\mathbf{f} = (\mathbf{M}\ddot{\mathbf{x}}) = -\xi_{\mathbf{G}} - \nabla\Phi - \mathbf{B}\dot{\mathbf{x}} \tag{2}$$

RMP-algebra introduces the `pushforward`, `pullback` and `resolve` operators used to construct the policy generation process. The goal of this process is to generate the desired accelerations $\ddot{\mathbf{q}}^d$ in the root node of the tree (robot configuration space) based on **q** and $\dot{\mathbf{q}}$. For further information about this framework, we refer the readers to the original paper [14].

Reinforcement Learning (RL) is a computational approach to automate policy learning by maximizing cumulative reward in an environment [33]. RL tasks are usually formulated as Markov Decision Processes (MDP). A finite-horizon, discounted MDP is characterized by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$, where the state and action spaces are respectively $\mathcal{S}$ and $\mathcal{A}$, transition dynamics $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, reward $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, an initial state distribution ρ_0, discount factor $\gamma \in [0, 1]$, and horizon T. The optimal policy $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$, maximizes the expected discounted reward:

$$J(\pi) = E_\pi\left[\sum_{t=0}^{T-1} \gamma r(s_t, a_t)\right] \tag{3}$$

4 Reactive Obstacle Avoidance Policies

Our main goal is to connect perception and motion in a single framework. Specifically, we aim to find the relationship between the visual input and the structure of the robot's motion. Intuitively, having such a relationship would mean that we can map visual data to information about the structure of the robot's motion in its environment. The explicit way to achieve this goal is to learn a mapping from raw images to the parameters ζ of the corresponding Riemannian metric $\mathbf{M}_\zeta(\mathbf{x}, \dot{\mathbf{x}})$. However, due to the large number of parameters in ζ, training such as mapping based on DRL would require a lot of samples. Instead, we consider an implicit method to achieve the same goal. Namely, we present a framework to learn vision-based reactive RMPs for obstacle avoidance. We design the action space and reward function in a way that encourages learning a similar but implicit representation of the vision-motion relationship. As modern machine learning methods depend on large datasets, robot learning in the real world can be time and resource expensive. In our method, we approach this problem from two different perspectives. First, we split the policy into two main parts. The first one is a hand-crafted **baseline policy**. The aim of this part is to guide the motion generation to achieve the task at hand. In our experiments, we consider goal-reaching tasks for which this

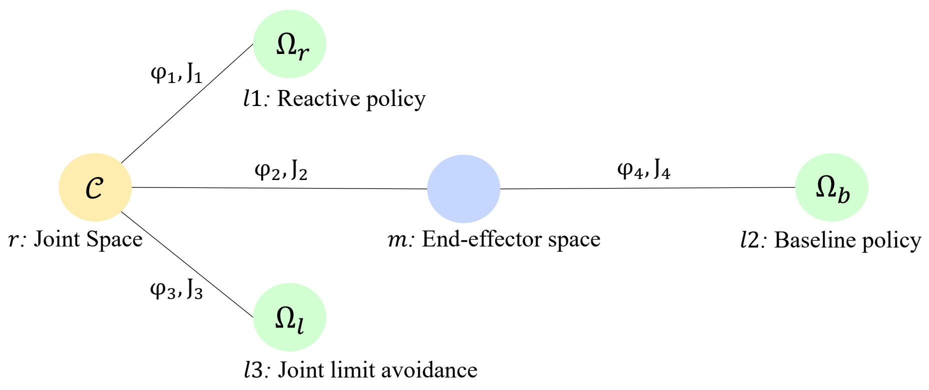


Figure 2: Structure of our RMP tree for combining multiple policies.

policy can be based on simple principles such as point attractors. We then augment this strategy by a **reactive policy** which can correct the baseline behavior depending on the perceived situation. This policy reacts to obstacles based on its knowledge of the end-effector position, current robot configuration as well as visual input. Instead of learning it in an end-to-end fashion, we initially pre-train our visual **latent model** based on data from random environment interactions. We then use the obtained latent representations as input to the reactive policy. Both, the baseline and reactive policies are formulated as RMPs, which allows us to effectively combine them using RMPflow. The full system is shown in figure 1.

4.1 Latent modeling

For representation learning we first collect images from our environment by sampling actions based on brownian motion [34]. The primary information we hope to obtain in the latent representations would correspond to object poses, shape and geometric information of the objects in the environment. To achieve that, we reduce the episode length during the data collection phase. Since obstacle types and poses are random at different episodes, having shorter episodes would result in more episodes for the same amount of data and hence more diverse obstacles and obstacle poses in the collected samples. Once data is available, we train a β Variational Autoencoder (β-VAE) [35] to obtain latent representations of our images. The objective function to train a β-VAE is the following:

$$\mathcal{L}(\theta, \phi, \mathbf{o}, \mathbf{z}, \beta) = E_{q_\phi(\mathbf{z}|\mathbf{o})}[\log p_\theta(\mathbf{o}|\mathbf{z})] - \beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{o})||p(\mathbf{z})) \tag{4}$$

where $\mathbf{o}$ corresponds to images in our case, $\mathbf{z}$ is the latent code, ϕ and θ are parameters of the encoder and decoder networks respectively, and β is a regularization coefficient. Higher β values help in learning more independent latent variables and hence improve disentanglement. It does so by forcing a stronger constraint on the latent bottleneck. Setting $\beta = 1$ would then reduce this approach to a standard VAE [36]. In our work, we use a β-scheduler based on [37].

4.2 Baseline policy

As previously mentioned, our primary task is goal reaching. For this purpose, we define a simple baseline RMP $\mathcal{R}_b = (\mathbf{f}_b, \mathbf{M}_b)_{\Omega_b}$ for goal reaching based on [9, 14], where Ω_b is a 3-dimensional manifold in which the baseline policy is defined. Namely, for $\mathbf{d} \in \Omega_b$, $\mathbf{d}$ measures the distance between the end-effector and the goal along the x, y and z axes. Specifically, we use a GDS to define the RMP in Ω_b. We first define the metric $\mathbf{G}_b$ to be an identity matrix, which eliminates the curvature terms in the GDS and to $\mathbf{M}_b := \mathbf{I}$ (based on the definitions in section 3), where $\mathbf{I}$ is the identity matrix. To ensure stability, we define the damping matrix as $\mathbf{B}_b := w\mathbf{I}$, where w is a vector that consists of positive constants close in value to zero. Moreover, we define the gradient of the potential field $\nabla\Phi(\mathbf{d}) := \mathbf{d}$. Substituting these values in equation (2), we obtain the GDS with the form

$$\mathbf{I}\ddot{\mathbf{d}} = -\mathbf{d} - w\mathbf{I}\dot{\mathbf{d}}. \tag{5}$$

This would result in the forcing term $\mathbf{f}_b = -\mathbf{d} - w\mathbf{I}\dot{\mathbf{d}}$ according to equation 2.

4.3 Reactive policy

To enable the obstacle avoidance behavior, we define a reactive RMP $\mathcal{R}_r = (\mathbf{f}_r, \mathbf{M}_r)_{\Omega_r}$ in a 7-dimensional manifold Ω_r. For $\mathbf{d}_q \in \mathbb{R}^7$ in Ω_r, $\mathbf{d}_q$ measures the distance between the current joint configuration $\mathbf{q}$ and the desired joint configuration $\mathbf{q}_g$. We set the metric, the damping matrix and the potential field of this RMP similar to the ones in the previous section. According to equation 2, the resulting forcing term is:

$$\mathbf{f}_r = -\mathbf{d}_q - w\mathbf{I}\dot{\mathbf{d}}_q. \tag{6}$$

We are then concerned in learning a mapping π_ψ from an input state $s_r = [\mathbf{q}, \dot{\mathbf{q}}, \mathbf{x}, \dot{\mathbf{x}}, \mathbf{z}]$ to an output action $a = \mathbf{d}_q$, where $\mathbf{x}$ corresponds to the end-effector position, and is obtained from $\mathbf{q}$ using the kinematics ϕ_1 of the manipulator, and $\dot{\mathbf{x}}$ is obtained from $\dot{\mathbf{q}}$ via the Jacobian $\mathbf{J}_1$. As for $\mathbf{z}$, it is the latent representation of a given image $\mathbf{o}$ (as in section 4.1). We model π_ψ as a multilayer perceptron (MLP) with two hidden layers. It is then trained via residual RL [30] with the baseline policy from section 4.2. For this purpose we use the Twin Delayed Deep Deterministic policy gradient algorithm [38]. Given the end-effector position $\mathbf{x}$, the goal position $\mathbf{x}_g$ and the action a per step, the reward is defined as follows:

$$r := r_{collide} + r_{goal} + r_{dist} + r_{control}, \tag{7}$$

where $r_{collide}$ is a negative reward of -10, which punishes the robot when colliding with an obstacle, r_{goal} is a positive incentive of 5 given only when the robot's end-effector reaches the goal, $r_{dist} = -1.6\|\mathbf{x} - \mathbf{x}_g\| + 0.75$ is the distance reward which rises linearly when the distance between the end-effector and the goal decreases, and $r_{control} = -0.05\|a\|$ punishes large actions. This last term encourages the policy to diverge from the baseline policy only in cases of possible collision. The framework is not sensitive to changes in the values used in the reward function as long as the general relation between the terms is preserved. Namely, the distance reward is an important signal for training. When its magnitude is large, it leads to a local solution where the collision reward is neglected. Additionally, we aim to ensure that the episode rewards for goal-reaching and collision avoidance are not dominated by the control reward, hence the small weight 0.05. This combination of rewards encourages the resulting policy to move along the geodesic of the corresponding Riemannian Manifold. Furthermore, our MDP has a finite horizon with episodes ending at a specified maximum number of steps or whenever the robot collides with an obstacle. During early exploration stages, most episodes are unsuccessful and end up with collision. Consequently, only few successful trials are recorded. We use Prioritized Experience Replay [39] in order to use data from such trials more efficiently.

4.4 Putting it all together

In addition to the baseline and reactive policies defined in the previous sections, we use an additional RMP $\mathcal{R}_l = (\mathbf{f}_l, \mathbf{M}_l)_{\Omega_l}$ for joint limit avoidance similar to the one in [9]. All three policies are then combined together based on the tree in figure 2. The node l_1 uses the RMP $\mathcal{R}_r$ from section 4.3, l_2 uses the $\mathcal{R}_b$ from section 4.2 and l_3 uses the joint limit avoidance RMP $\mathcal{R}_l$. Consequently, RMP-algebra is applied to compute the desired joint acceleration $\ddot{\mathbf{q}}_d$.

To ensure safety, we use impedance control with suitable collision handling algorithms guaranteeing contact force and torque thresholds, enabling quick collision detection and safe reaction [1]. At the RL level, when a collision occurs during task execution, it is detected safely and early to prevent damage (based on [1]), the episode terminates and the environment returns a negative reward as to inhibit such behavior in the future.

5 Experiments

We conduct experiments to answer the following questions: (i) is our method capable of learning successful obstacle avoidance strategies? (ii) how does the baseline policy contribute to the overall sample-efficiency? (iii) is our approach capable of simultaneously avoiding multiple obstacles?

5.1 Setup

All of our experiments share the same setup. Namely, we use a seven DoF Franka Emika Panda robot in a Gazebo simulation environment [40]. We place a static RGB camera in front of the robot

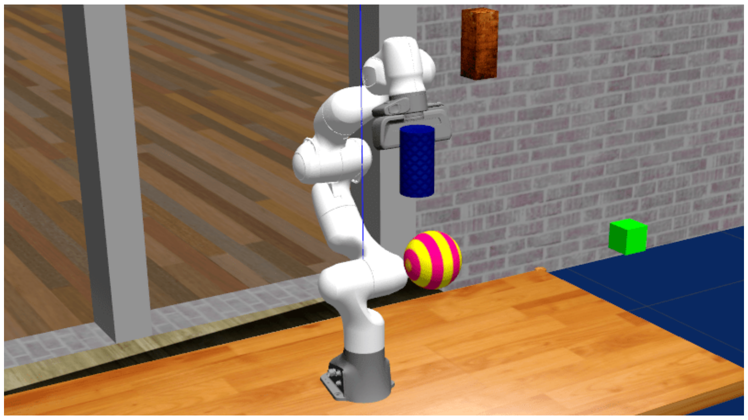


Figure 3: Illustration of our task setup in simulation.

(green block in figure 3). We use three different obstacle shapes: cuboids, spheres and cylinders. Each obstacle has a distinct texture. In our experiments, we sample the object positions to be true obstacles with respect to the goal. The environment setup is illustrated in figure 3. Furthermore, we use an NVIDIA GeForce RTX 2080 GPU to train our vision module, and reactive policy. For all of our experiments, we downsample the RGB images to 128×128. All of our results are averaged over 5 training trials with different random seeds. We look at the success rate and the average episode return (AER). For improved visibility, we smoothen the AER plots using running means with a window size of 20 episodes. For measuring the success rate, we label a trial as successful, when the end-effector reaches the desired goal with no collision and manages to stay there for 5 seconds.

5.2 Results

Experiment A. In our first experiment, we want to evaluate our method on a single obstacle avoidance task. Additionally, we want to see the effect of the baseline policy on the overall performance. We refer to the case without a baseline policy as **Vanilla Learning (VL)** and the one with baseline policy as **Residual Policy Learning (RPL)**. VL would then correspond to a standard DRL approach for obstacle avoidance with the addition of the latent model and the RMP handling of $a = \mathbf{d}_q$. In figure 4, we compare the average episode reward over time (left) and success rate (right) of both possibilities. After approximately 80 min, both policies converge and the average reward as well as the success rates remain more or less the same. The total reward accumulated using RPL is substantially higher than that of VL. The latter manages to avoid the obstacle in few trials, however, it fails in most cases to also reach the goal, or even get close to it. This behavior can be shown by looking at the individual reward term from equation (7). We provide a plot of these values in the supplementary material. Furthermore, vanilla learning requires at least twice the amount of samples to reach the same success rate as our method. This is mainly due to two factors. First, for our use case, VL attempts to learn both obstacle avoidance and goal reaching. However, in RPL, goal reaching is taken care of by the baseline policy. One could argue that providing a baseline policy limits the flexibility of the model, which usually leads to declined performance. However, our experiments show a different tendency. Even after convergence, our method achieves more than twice the success rate ($84 \pm 6\%$) than the vanilla method ($39 \pm 27\%$), which seems to often get stuck in a local solution (it reaches a lower average reward even after convergence). The second reason for this difference in sample-efficiency is exploration. In its early stages, our method already tries to reach the goal, as it is guided to do so by the baseline policy. While trying to do this, it also encounters the obstacles more frequently and receives negative reward. However, in the absence of a baseline, the exploration is random, and results mostly in low-reward trajectories for training.

Experiment B. In the second experiment, we aim to test the capability of our method to learn to avoid multiple obstacles simultaneously. The only difference is that we train the policy in an environment containing 3 obstacles at all times (as shown in figure 3). As expected, the amount of interactions needed for this task to be learned is substantially higher than that of the single obstacle case. The policy converges after 10000 episodes which is equivalent to ~ 3 hours of data collection with a single robot. The success rate after convergence is 72%. We show the evolution of the AER

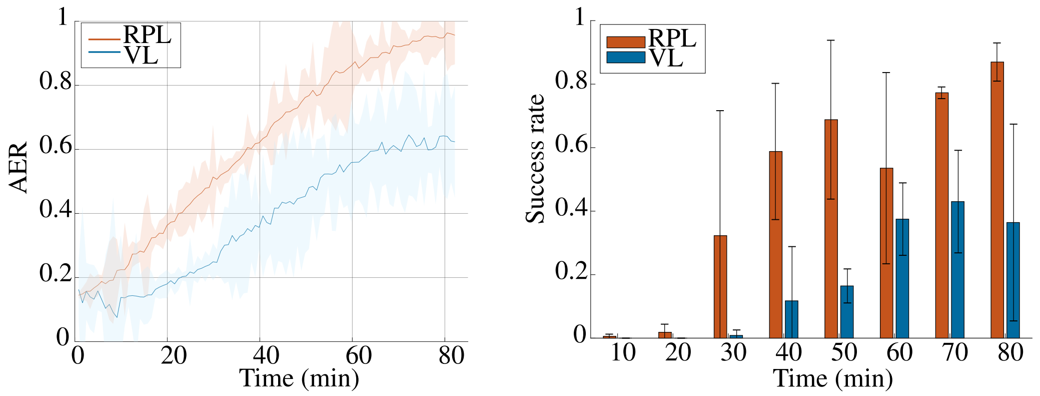
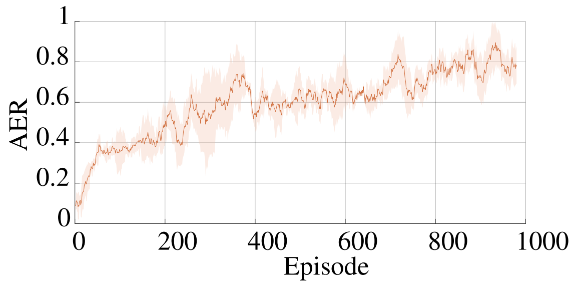


Figure 4: **Experiment A**. Effect of the baseline policy on the single obstacle avoidance performance. (Left) Normalized average episode reward (AER) over time. (Right) Success rate over time.



# obstacles	Success	Near Goal	Failure
1	66%	30%	4%
2	54%	44%	2%
3	72%	22%	6%

Figure 5: **Experiment B** Multiple obstacle avoidance. (Left) Normalized average episode reward (AER). (Right) Generalization of a policy trained for 3 obstacles on scenarios with 1 and 2 obstacles.

over time in figure 5 (left). Furthermore, we test the obtained policy on environments containing 1 and 2 obstacles to check whether the learned reactive behavior can generalize to different scenarios or just memorizes sequences of actions depending on the obstacles configuration. We evaluate our policy for 50 trials for each scenario. We report **success** as previously defined, **near goal** reaching which corresponds to avoiding the obstacle and getting to the close proximity of the goal ($\sim 7cm$) but not exactly reaching it, and **failure** which corresponds to all other cases. The results are shown in figure 5 (right). Interestingly, even when only trained in 3 obstacles environments, the obtained policy still manages to succeed in reaching the goal or its near proximity at a high rate, even when tested on environments with 1 and 2 obstacles. These results strongly support our claim for generalization.

6 Conclusion

We propose a unified framework for learning vision-based obstacle avoidance strategies for robot manipulators. The presented approach treats perception and planning simultaneously and formulates the problem as a reinforcement learning problem in Riemannian Manifolds, successfully connecting perception and motion together. Our approach takes advantage of the latest advances in motion primitives as well as representation learning, to improve on the sample-efficiency and the safety of exploration aspects of learning. Based on that, we learn successful reactive policies, capable of avoiding both single and multiple obstacles. In addition, our experiments demonstrate the feasibility and sample-efficiency of our method. Finally, our framework is limited to static environments, we aim to address this issue in future work.

Acknowledgments

We greatly acknowledge the funding of this work by Vodafone AG, Microsoft Germany and the Alfried Krupp von Bohlen und Halbach Foundation. Please note that S. Haddadin has a potential conflict of interest as shareholder of Franka Emika GmbH.

References

[1] S. Haddadin, A. Albu-Schaffer, A. De Luca, and G. Hirzinger. Collision detection and reaction: A contribution to safe physical human-robot interaction. In *International Conference on Intelligent Robots and Systems*. IEEE, 2008.

[2] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. G. Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg. Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*, 2018.

[3] A. Rai, G. Sutanto, S. Schaal, and F. Meier. Learning feedback terms for reactive planning and control. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[4] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox. Deepim: Deep iterative matching for 6d pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[5] K. Park, T. Patten, and M. Vincze. Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[6] M. Pecka and T. Svoboda. Safe exploration techniques for reinforcement learning–an overview. In *International Workshop on Modelling and Simulation for Autonomous Systems*. Springer, 2014.

[7] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

[8] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. *arXiv preprint arXiv:1906.08880*, 2019.

[9] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

[10] S. M. Khansari-Zadeh and A. Billard. A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots*, 2012.

[11] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*. Springer, 1986.

[12] S. Haddadin, H. Urbanek, S. Parusel, D. Burschka, J. Roßmann, A. Albu-Schäffer, and G. Hirzinger. Real-time reactive motion generation based on variable attractor dynamics and shaped velocities. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010.

[13] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 1987.

[14] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff. Rmpflow: A computational graph for automatic motion policy generation. In *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2018.

[15] N. Ratliff, M. Toussaint, and S. Schaal. Understanding the geometry of workspace obstacles in motion optimization. In *International Conference on Robotics and Automation*. IEEE, 2015.

[16] M. Watterson, S. Liu, K. Sun, T. Smith, and V. Kumar. Trajectory optimization on manifolds with applications to so (3) and r3xs2. In *Robotics: Science and Systems*, 2018.

[17] M. S. Guzel and R. Bicker. Vision based obstacle avoidance techniques. *Recent advances in mobile robotics*, 2011.

[18] S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*. Springer, 2006.

[19] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, 2013.

[20] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on control systems technology*, 2009.

[21] Abraham, R. Cuautle, M. Osorio, and R. Zapata. *Reactive Motion Planning for Mobile Robots*. 2008.

[22] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 2002.

[23] C. Bodnar, A. Li, K. Hausman, P. Pastor, and M. Kalakrishnan. Quantile qt-opt for risk-aware vision-based robotic grasping. *arXiv preprint arXiv:1910.02787*, 2019.

[24] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[25] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[26] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 2018.

[27] J. Lin, H. Zhu, and J. Alonso-Mora. Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments. *arXiv preprint arXiv:2002.04920*, 2020.

[28] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra. Deep reinforcement learning for collision avoidance of robotic manipulators. In *European Control Conference (ECC)*. IEEE, 2018.

[29] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[30] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[31] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 2016.

[32] B. Kim, C. Jang, and J. Hong. Geometric algorithms for robot dynamics: A tutorial review.

[33] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[34] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 1930.

[35] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2016.

[36] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[37] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[38] S. Fujimoto, H. Van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[39] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[40] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2004.

A Implementation Details

Previous work has shown that the performance of reinforcement learning algorithms is dependent on the implementation details [1, 2]. Thus, we provide further details about our method's models and training procedures for both representation (section A.1) and reinforcement learning (section A.2) for the sake of reproducibility.

A.1 Representation Learning

Our β-VAE encoder is a convolutional neural network (CNN) with five convolutional layers having in order 6, 32, 64, 128 and 256 channels. Each layer uses a rectified linear unit (ReLU) for activation and is followed by a batch normalization layer [3]. The mean and log standard deviation layers of the latent are linear and are also followed by a batch normalization layer. The mean layer uses a hyperbolic tangent (Tanh) activation.

For training, we use the Adam optimizer [4] with a learning rate of 0.001 and batches of size 128.

A.2 Reinforcement Learning

TD3 is an actor-critic method [5]. We use an actor with two hidden layers, each containing 100 neurons. The activation function of all layers is Tanh. The critic has two hidden layers, with 500 neurons each. For the critic, we use ReLU activations for hidden layers and no activation for the output. We use Adam to optimize both sets of parameters. The training hyperparameters are listed in the table below.

Hyperparameter	**Value**
TD3 Policy noise	0.2
Max episode steps	400
Exploration noise	$0.5 \rightarrow 0.3$
Memory Size	300000
Batch size	64
Learning rate	$1e-3$

B Simulation-Reality Gap

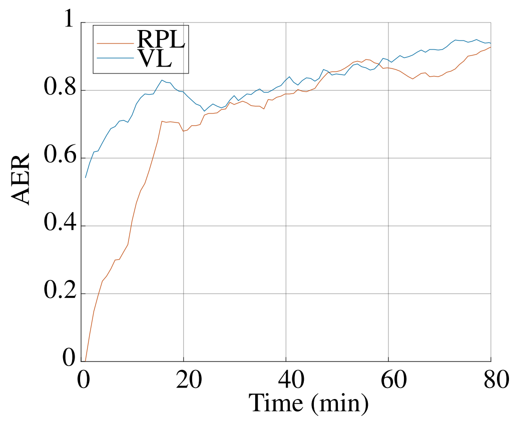
We argue that the Simulation-to-Reality gap is not problematic for our experiments design. This is due to the nature of our task and goal, being concerned with pure geometric motion and not with dynamics, the latter being a lot harder to simulate. In our previous experiences, the main observed difference between simulation and the real-world, for vision-based motion generation tasks is in the noise distributions of images. This is due to mainly two factors. The first one is based on the lack of photo-realistic rendering in simulation environments. The second is due to the use of over-simplified environments with compact color backgrounds and minimum clutter. We compensate for this last factor by adding textures to the obstacles and by replicating the setting of common robot learning labs. Furthermore, VAE-based methods have been shown to capture real-world images well [6, 7]. We use a β-VAE for our visual latent model, which would make the transition to real-world images only as expensive as the data collection process.

C Additional results

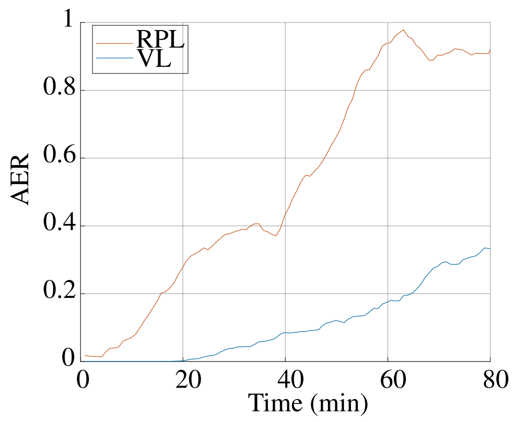
In this section we show some additional results related to experiments A and B (section C.1 and C.2) as well as to the latent model (section C.4).

C.1 Experiment A

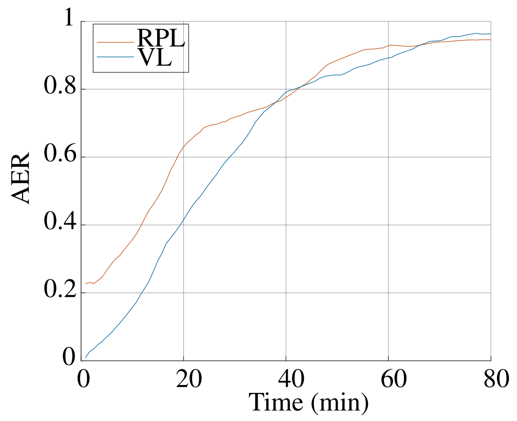
In addition to the previously shown total reward plot, here we show plots of the individual reward terms: $r_{collide}$, r_{goal}, r_{dist}, $r_{control}$. These values are especially interesting when comparing the policy learning with the baseline and without it. As before, we refer to those cases as **Residual**



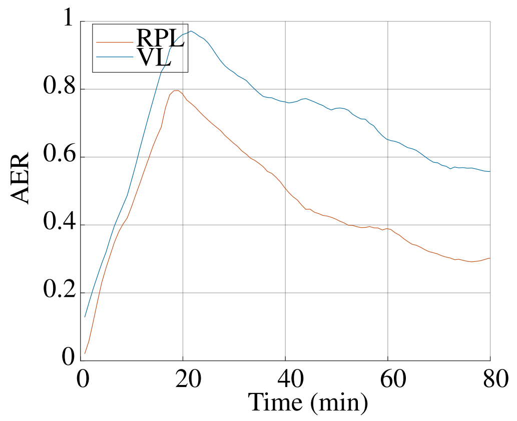
(a) $r_{collide}$



(b) r_{goal}



(c) r_{dist}



(d) r_{ctrl}

Figure 1: Individual rewards for the policy training with a baseline (RPL) and without (VL).

Policy Learning (RPL) and **Vanilla Learning (VL)** respectively. The results can be seen in figure 1. As expected the goal reward for RPL is substantially higher than that of VL at all times. At the beginning of training, RPL leads to more collisions with the obstacle as the baseline policy guides it towards the goal. Subsequently, RPL starts with more negative reward $r_{collide}$ than VL. However, it manages after training to reach a similar level as VL. In contrast, the latter depending mostly on random actions barely hits the obstacle at these stages as it's not even directed towards the goal. This can be seen in the r_{goal} and r_{dist} plots. As for the control reward r_{ctrl}, it behaves similarly for both methods. However, it gets higher for vanilla learning after a while. This could be explained by the following: RPL based exploration leads at all times to higher goal reward than VL. The latter, barely reaching the goal, prefers to take smaller actions to increase r_{ctrl}.

C.2 Experiment B

In addition to the provided video here we show image sequences for our multiple obstacle avoidance results. We show a successful trial in figure 2 and an unsuccessful trial in figure 3.

C.3 Experiment C

Although experiment B supports our claim of generalization, we conduct a further experiment to double-check our method's generalization ability. This experiment tests if the trained policy can generalize to unseen obstacles. We train our policy in single obstacle avoidance scenarios, with two types of obstacles: cuboid and sphere. After the training, we test the trained policy in scenarios with a cylinder obstacle. We evaluate the policy for 50 trials in the scenarios containing the unseen

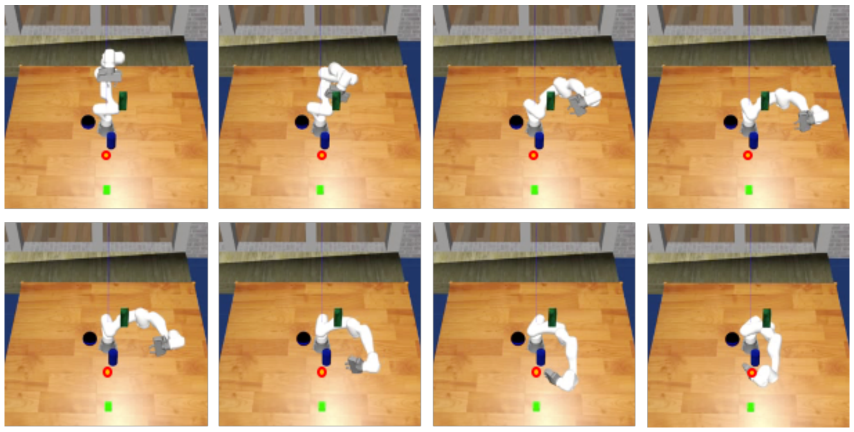


Figure 2: Multiple Obstacle Avoidance: Successful trial. The red circle is the goal.

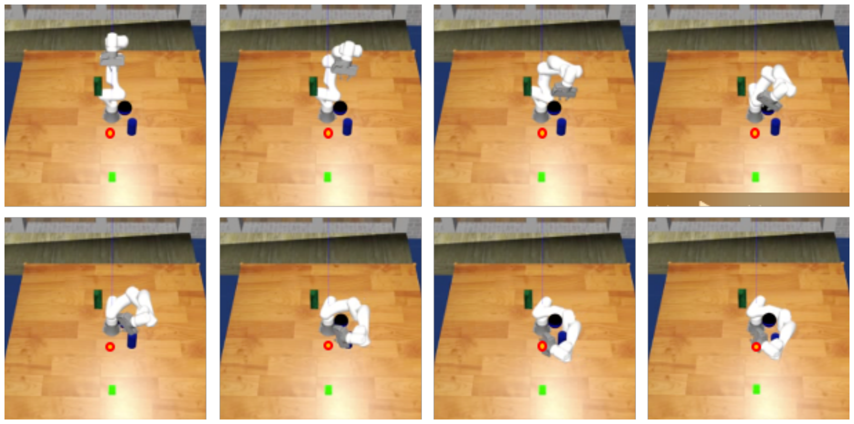


Figure 3: Multiple Obstacle Avoidance: Unsuccessful Trial. The red circle is the goal. The robot collides with the cylinder at the end of the execution.

obstacle. We report that the success rate is 72%, which is similar to the success rate when evaluating the policy using the previously-seen obstacles. Besides the training curve, we also provide an image sequence of the execution. Figure 4 and 5 are evaluations of the policy on trained obstacles (cube and sphere). Figure 6 shows a policy execution in the scenario with previously unseen obstacle (cylinder). The result of this experiment provides another evidence of the generalization ability of our method.

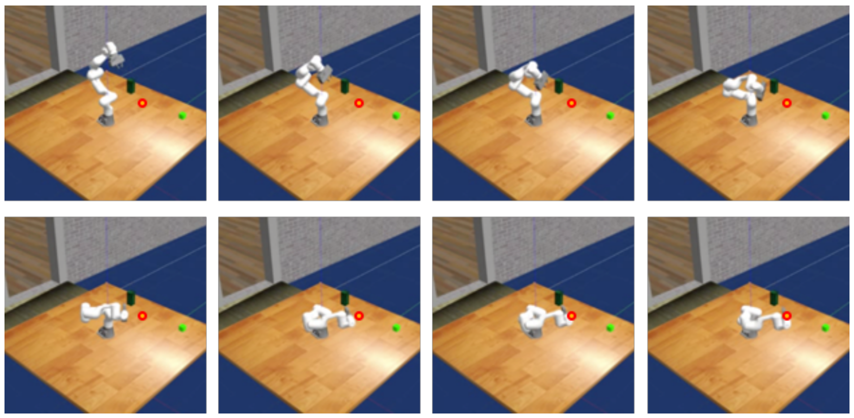


Figure 4: Single obstacle avoidance: cube. The red circle is the goal.

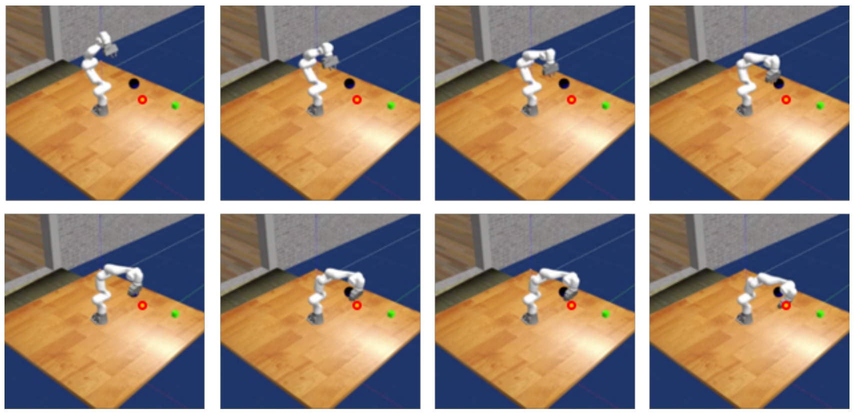


Figure 5: Single obstacle avoidance: sphere. The red circle is the goal.

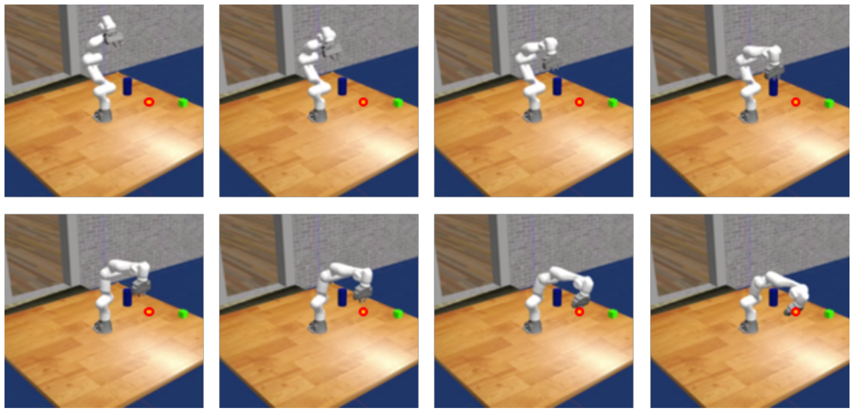


Figure 6: Single obstacle avoidance: cylinder. The red circle is the goal.

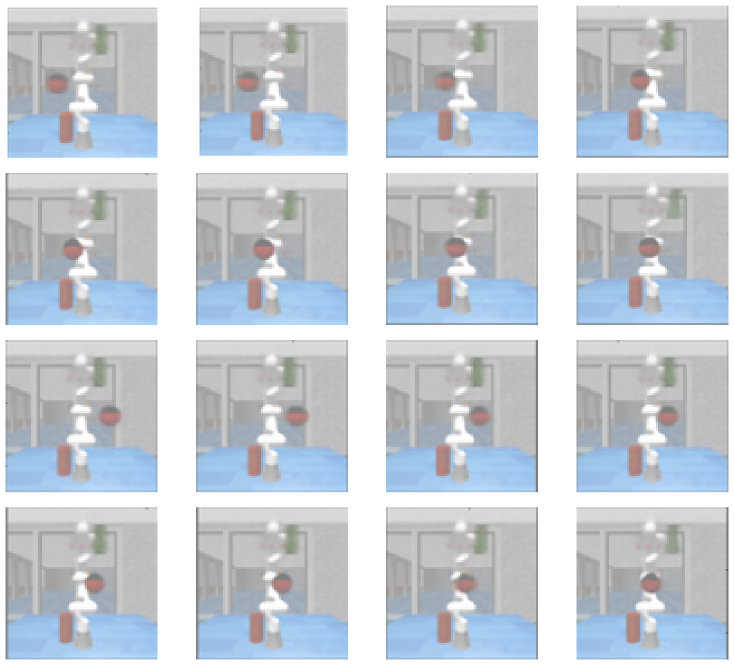


Figure 7: Images Sampled from latent z_0 and z_1. First two rows: $z_1 < 0$ and $z_0 \in [-1, 1]$; Last two rows, $z_1 > 0$ and $z_0 \in [-1, 1]$. z_0 and z_1 control together the y-position of the sphere.

C.4 Sampling the latent

Here we illustrate images sampled from our VAE's latent as to show the importance of such variables to the obstacle avoidance task. The samples are shown in figure 7. Note that the sampled images are only blurry because of the down-sampling of the inputs images to the VAE.

References

[1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI*, 2018.

[2] N. Rao, E. Aljalbout, A. Sauer, and S. Haddadin. How to make deep rl work in practice. *arXiv preprint arXiv:2010.13083*, 2020.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

[6] H. Van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *International conference on intelligent robots and systems (IROS)*. IEEE, 2016.

[7] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.